Role Recommender-RBAC: Optimizing User-Role Assignments in RBAC

K. Rajesh Rao[a,e], Ashalatha Nayak[b,*], Indranil Ghosh Ray[c], Yogachandran Rahulamathavan[d], Muttukrishnan Rajarajan[e]

[a]*Department of Information and Communication Technology, Manipal Institute of Technology, MAHE, Karnataka, INDIA.*
[b]*Department of Computer Science and Engineering, Manipal Institute of Technology, MAHE, Karnataka, INDIA.*
[c]*Department of Computer Science, University of Warwick, ENGLAND.*
[d]*Institute for Digital Technologies, Loughborough University London, ENGLAND.*
[e]*Department of Electrical and Computer Engineering, City, University of London, London, ENGLAND.*

Abstract

In a rapidly changing IT environment, access to the resources involved in various projects might change randomly based on the role-based access control (RBAC) system. Hence, the security administrator needs to dynamically maintain the role assignments to users for optimizing user-role assignments. The manual updation of user-role assignments is prone to error and increases administrative workload. Therefore, a role recommendation model is introduced for the RBAC system to optimize user-role assignments based on user behaviour patterns. It is shown that the model automatically revokes and refurbishes the user-role assignments by observing user access behaviour. This model is used in the cloud for providing Role-Assignment-as-a-Service to optimize the cost of built-in roles. Several experiments are conducted to verify the proposed model using the Amazon access sample dataset. The experimental results show that the efficiency of the proposed model is 50% higher than the state-of-the-art.

Keywords: Access control, Cloud computing, Hidden Markov model, RBAC, Role recommendation

1. Introduction

To manage legitimate access to data, functionality, services, and resources, researchers employ various access control policies. At present, role-based access control (RBAC) is the most preferred choice of many leading organizations due to minimal overhead when user assigned permissions are frequently analyzed and updated. The idea of RBAC [1] lies in creating roles, which further incorporates a set of permissions. Each role is suitable for a particular task, and thus all permissions necessary to carry this task are known as permission-role assignments. The users are then assigned a set of roles known as user-role assignments depending on the tasks to be performed. Figure 1 illustrates user-role assignments (UA) and permission-role assignments (PA) in RBAC for completing a particular task by the user. Role engineering, particularly *role mining*, is the most widely used and preferred choice of researchers for successful RBAC adoption in any enterprise [2].

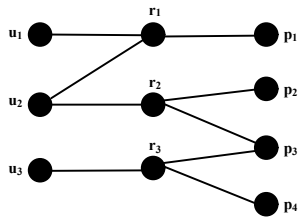

Figure 1: Users access permissions through roles

In RBAC, due to the dynamic nature of the business processes, users may access only a limited number of roles from the set of assigned roles. In such a case, role

*Corresponding author
Email address: asha.nayak@manipal.edu (Ashalatha Nayak)

assignments for users may turn obsolete, which means that the corresponding UA is not longer required at this point of time [2]. Similarly, role assignments for users may need to be refurbished, i.e., they might be obsolete at some time in the past, but are required at this point of time. Therein lies the biggest problem of RBAC as to how exactly a security administrator can dynamically maintain role assignments by optimizing the UA? Since the main principle of RBAC lies in providing the least privilege principle (LPP) [3], which means that a user is not given more privileges (i.e., role) than necessary to perform a job. In role mining [4, 5] and role refinement [6, 7] techniques, the authors have optimized the UA by refining the existing role set. Such approaches are used to optimize the UA at the system level, i.e., the role requirement is checked among all the users. Therefore, by determining whether to add or delete the UA at the system level does not necessarily optimize the UA at the user level.

As far as is known, to date, no role recommendation model is available, which is integrated with RBAC to optimize the UA at the user level. In this paper, the following three real-world scenarios are discussed and analyzed where the proposed system is a good choice for commercial applications. The proposed system was developed using the Amazon access sample dataset [8], and successfully optimized the UA based on the recently accessed permissions. The performance of the proposed system was also evaluated by comparing it with the existing models using the synthetic datasets proposed in [9].

Scenario 1: *User A is involved in various projects of an organization. Due to the completion of the project or other reasons, user A may not use a certain set of roles and the corresponding UA now becomes obsolete.*
Scenario 2: *When user A wants to access a certain set of roles, which were revoked at some time in the past but is now required, then the corresponding UA needs to be refurbished by considering it as an emergency requirement.*
Scenario 3: *User A is given privileges for a certain period of time, after which the user needs to request for extension of his privileges.*

In leading cloud platforms such as Microsoft Azure, the need of optimizing UA in deploying the applications was investigated. Consider Azure resource manager [10], which provides fine-grained access control of resources using a built-in RBAC system. Seventy built-in roles can be assigned to users to provide restricted access. Such predefined roles are assigned with the available constant number of resources that cannot be customized. For example, in Azure, there exist roles such as *Monitoring Reader* and *Log Analytics Contributor*, which can access and monitor all the log activities, i.e., user behaviour/access pattern. Hence, many organizations use these built-in roles while deploying the RBAC enabled applications in Azure Cloud. Meanwhile, organizations need to pay for the cloud service provider (CSP) based on the usage of roles, rather than the usage of permissions. Due to the dynamic nature of business processes, users may not use all the resources/permissions, which are assigned through the roles. Since the PA cannot be altered, the only option to make effective pricing/utility computing for organizations is to optimize the UA based on the roles.

In this paper, a solution is proposed to optimize the UA in the RBAC system. The solutions are presented in two phases: 1) extraction of user behaviour pattern from the system usage log data, and 2) matching the users from this pattern to recommend the roles by checking whether the UA is obsolete or whether the UA needs to be retained or refurbished. The Hidden Markov model (HMM) represented as a dynamic Bayesian network was used to carry out the role recommendation. This is said to be role recommender, which recommends the roles aligned with the user privileges to optimize the UA, which can be done by integrating with the RBAC. Here, it is represent it as *Role recommender-RBAC (R-RBAC)* on this paper. For example, in RBAC enabled applications, roles as well as its assignments to users and permissions have a predefined period beyond which it expires. However, for some users, it may not be necessary to have such role assignments longer than required. Therefore, the proposed system monitors the access pattern of the users and automatically revokes the UA before the fixed deadline, rather than waiting for the deadline or for manual revoking operation by the security administrator. The refined UA by R-RBAC in the cloud applications optimizes the built-in roles assigned to the users, which in turn provides cost optimization for organizations. The services provided on hosting R-RBAC in cloud applications is referred to as Role-Assignment-as-a-Service (RAaaS).

The optimal solution in UA means the roles assigned to the user must satisfy the LPP. Since optimizing UA by

maintaining the LPP is proved to be NP-hard [6], a good approximation is provided to the optimal solution in contrast to the state-of-the-art. Updating the UA by role recommendation has a direct impact on meeting the principle of least privilege. So, the user as well as the security administrator will always have a privilege to request a new set of recommended roles by the R-RBAC, if the roles assigned to the user does not satisfy the principle of least privilege.

Based on the above discussions, the following specific contributions are listed out.

1. Role recommendation model for the RBAC system (R-RBAC) recommends the roles to optimize the UA by revoking obsolete UA, and refurbishing and retaining necessary UA based on user behaviour pattern. It is shown that the R-RBAC recommends the roles on Amazon access sample dataset [8] to optimize the UA by retaining an average of 57% of roles when 50% of UA is obsolete, and restoring an average of 28% of roles when 10% of UA needs refurbishment. The consequences of such optimization helps cloud applications to optimize the cost of built-in roles, which is referred to as Role-Assignment-as-a-Service.

2. The R-RBAC recommends the roles for each user to optimize the UA at the user-level, unlike the existing models which optimize the UA at the system level. The efficiency of the R-RBAC on *size* and *speed* metrics can be seen up to 32% and 50% respectively, when compared with Least Privilege User-Role Assignment Problem [6] using the synthetic datasets.

The rest of the paper is organized as follows: Section 2 discuss the related works. In Section 3, the proposed methodology is provided. In Section 4, the R-RBAC analysis is presented. In Section 5, the experimental results are shown, and finally, Section 6 concludes the paper.

2. Related Work

Recently many approaches have been identified to approximate the optimal solution for role generation and role assignments in access control systems such as role mining, dynamic user-role assignments, and role refinement. Such approaches are not suitable to optimize the UA when the user privileges need dynamic changes, for example, the three scenarios described in the introduction section. The following sections shall provide the details of the mentioned approaches.

2.1. Role Mining Methods

Role engineering, especially role mining, is the most widely used and preferred choice of researchers during the implementation of RBAC to maintain LPP by optimizing the UA as well as PA [2]. In most of the role mining techniques, for deriving role assignments to users as well as permissions, the available user-permission assignments are taken into consideration. The first proposed technique for role mining is based on the initial clustering of users, who have been assigned the same permissions [11]. Basic-RMP [4] finds a minimal set of roles from user-permission assignments and provides role assignments to users as well as permissions. Researchers have found that a feasible set of roles can be obtained to optimize user access by mapping the role mining problem (RMP) to well-known problems (NP-hard). A solution to such a kind of problem may be used for the RMP. For example, Basic-RMP is mapped to minimum tiling problem [4]- where each tile corresponds to a role, minimum biclique cover [12]- where each role corresponds to biclique, and set cover problem [6]- where each subset corresponds to a role. In edge-RMP [5], work has been done to minimize the administrative burden by optimizing user-role as well as permission-role assignments. Since Basic-RMP and edge-RMP proved to be NP-hard, a greedy and approximation algorithm was proposed to optimize the edges (i.e., UA and PA) in RBAC.

An unsupervised approach of role mining called Fast Miner [9], is based on the permission set enumeration on the predefined constraints. The Simple Role Mining Algorithm [13], is a heuristics-based solution to approximate the optimal role set. The user with the least number of permissions will be the initial entry to the role set. This process of selection on the least number of permissions is done progressively after completion of the individual user's task. The subsequent updation of the role set is maintained by eliminating the roles obtained as union of other roles which is already inserted in the role set. The HP Role Minimisation algorithm [12] and the Weighted Structural Complexity Optimization [14] are exact variants of RMP, as the set of roles is strongly compatible with

the permissions assigned to the users. The process of mining the roles are also incorporated in the RBAC extension models such as Temporal RBAC and Generalized Temporal RBAC. This is known as the Temporal RMP [15]. Here, the role assignments to the users and permissions are enabled only for a set of time intervals. In constraint role miner [16], a proposed role mining algorithm complies with various kinds of constraints for optimizing the role assignments to the users and permissions.

All the above role mining approaches maintain LPP by optimizing role assignments to the users as well as permissions, which are closely compatible with the user-permission assignments. After a certain period, if some UA becomes obsolete or needs refurbishment, then the security administrator can manually update such assignments. This will become a tedious task for the security administrator, if the updating is frequent and required for a large number of UA. To date, there is no user-level role recommendation model for the RBAC system to optimize the UA, dynamically based on the requirements. Further, in R-PEKS [17], it is proposed in a future work that there is need to optimize the UA for better performance and security in dynamic user-permission assignments.

2.2. Dynamic User-Role Assignments

In Rule-Based RBAC [18], the administrative unit of an enterprise maintains dynamic UA by considering the credentials of the user and imposes certain constraints on the roles. In TrustBAC [19], the roles are assigned dynamically to the users based on the trust levels maintained by the users during the past use of credential. Further, these trust components also include past behaviour and recommendations from other users. In Role-Based Trust Management Framework (RT) [20], for each request, the entire credential phase is repeated because the role assigned for the user is for a specified time interval. All the above dynamic UA may not be a good choice, where users access the permissions regularly.

Recently, in [21], RBAC was combined with trust and reputation for secure data access in the cloud. In such a mechanism, roles are assigned to the users based on experts' recommendations, role behaviour, and user behaviour analysis. This can be a solution to the problem of remote access control for maintaining UA dynamically when the users are not formerly known, i.e., unauthorized users. Further, RBAC is combined with attributes (i.e., RBAC-A) [22] such as time and location to make it dynamic in a large scale fast changing domain. Additionally, discriminative learning is used to generate the RBAC system. But such dynamic role assignments in RBAC do not recommend roles that may optimize the UA based on attributes.

2.3. Role Refinement

The refinement of an existing set of roles to meet the demands of new requirements that exist within the organization is said to be known as role refinement problem [7]. the ad hoc creation of roles to satisfy new requirements increases administrative overhead. Therefore, it is always recommended to periodically refine the roles in the existing role system. The role refinement problem is solved by creating a new system of roles (refined role system), i.e., by considering a candidate set of roles as input. Applying the role refinement is proved to be NP-hard, and a greedy and randomized approach solves the problem [7]. Here, role refinement also tries to optimize the UA at the system level by assigning a refined role set to the users. Further, during the role refinement process, the obsolete UA is not taken into consideration.

Optimization in UA is done by maintaining least privilege and is said to be the least privilege user-role assignment problem (LPUAP) [6]. Since it is proved to be NP-hard, the problem is solved using the approximation algorithm by adjusting the roles (sometimes called role refinement). Once the roles are assigned to the users, post-processing is done to reduce the number of roles. The roles can be deleted if other existing roles cover the corresponding permissions among all the assigned users by maintaining only the least privileges. While adding the roles, if the existing roles do not cover the permissions, then new roles are added. The model is more applicable only if a user does not use a particular role, or if a new role is added to the system. Therefore, the optimization of UA is solved at the system level, rather than at the user level. In [23], the process of security refinement is also available at the network access control. Here, the refinement is meant to obtain an optimal network security configuration by the placement of the security devices based on exploring different design alternatives.

The existing research suggested an idea of using historical logs to refine the original role system [2]. This is a very closed idea to maintain optimization in UA, where

obsolete UA may be analyzed from the historical logs. But the idea was just proposed in [2] and further, there exists a challenge based on the requirement as to how the system can refurbish the UA. Therefore, due to the dynamic nature of the business processes, there is a necessity to update the UA (by revoking obsolete UA as well as by refurbishing necessary UA) by maintaining the least privileges at the user level. During the UA optimization at the user level, the existing approaches such as role refinement cannot be incorporated as refining the existing roles will impact on the other users. One possible solution can be to dynamically assigning the roles to the user by maintaining the least set of privileges.

Based on the above limitations, the objective of the proposed approach is to develop a role recommendation model for the RBAC system (i.e., R-RBAC). Such a system could help to optimize the UA by recommending roles for each user, which are a good approximation to the optimal solution. For recommending the roles, the HMM was used, which is represented as a dynamic Bayesian network and integrates with the RBAC system.

To develop a recommendation model, it was decided to implement an HMM. Given a set of permissions requested and granted to the user, what are the chances that the user will be in one of the permission sets in the near future? Such uncertainty is based on the assumption that the user behaves in a similar manner when migrating between states [24]. For example, an employee who is assigned the same type of tasks will require the same set of permissions and carry out the same actions. Thus usage behaviour will be similar, and the current permissions requested can be implied from past permissions [25].

To generate an RBAC system, it was decided to use the Simple Role Mining (SRM) algorithm [13]. The two main criteria to consider while implementing role mining algorithm is the speed and quality of the produced role set. Speed is crucial as some algorithms grows exponentially depending on the number of users and permissions, and thus, are not suitable for large systems. There is of course a question of how to measure the quality of the roles. Some researchers focus only on the size of the generated role set, while others introduce the notion of weighted structural complexity [14], [26], which sums up the number of relationships in an RBAC state, with possibly different weights for each relationship. The HP Role Minimisation algorithm [12] performs well using both of the metrics mentioned above [26, 27]. SRM [13] has taken this as a base and applied a heuristic to simplify the algorithm. Instead of looking at how many users are covered under each new role, they pick the role that contains the smallest number of permissions. It turns out that this approach works well, and moreover is faster than the HP algorithm and produces a role set whose quality is close to the one produced by the HP algorithm. For these reasons, it was chosen to implement the SRM algorithm.

A comparison of the proposed work with five related works can be seen in Table 1. Along with the LPP, the proposed approach also provides cost optimization while using built-in roles in the cloud platform. In the proposed approach, speed and percentage of optimization are relatively better because the UA optimization is performed at the user level, unlike the existing models, which do it at the system level. Further, unlike the existing models, the proposed solution periodically identifies obsolete UA as well as refurbishes the necessary UA.

3. Proposed Methodology

In this section, the design of the R-RBAC system as well as its components to recommend the roles and to optimize the UA at the user level is proposed.

3.1. Design of R-RBAC System

The design of the role recommendation model for the RBAC system that optimizes the UA is shown in Figure 2. The RBAC consists of role assignments to the users as well as permissions. Users access the logger, which handles the assigned permissions based on the RBAC and such sequences of accessed permissions are segregated based on the user's identity and are stored in log files, as shown in Figure 2. In the log files, a set of accessed permissions and a set of last $'k'$ accessed permissions at time t from each user are further referred to as user behaviour pattern (BP) and current usage of user behaviour pattern (BP^k) respectively. The role recommender consists of HHM generator, user behaviour analyzer, and role matcher, which are used in the processes of recommending the roles to optimize the UA. The Recommended-RBAC contains the PA and the optimized UA. The solution is optimized by revoking obsolete UA, and retaining and refurbishing the necessary UA. Moreover, users continue to access the permissions based on

Table 1: Properties of different UA optimization techniques in RBAC

Property	**Role Mining algorithms [4, 5]**	**LPUAP [6]**	**Role refinement [7]**	**Dynamic UA [21]**	**Proposed approach**
Key concerns	LPP	LPP	LPP	LPP	LPP and cost optimization for organization
Optimization Technique	greedy and approximation	approximation	greedy and randomized	trust	HMM
Speed of optimizing a role assigned to users	depends on |UA| and |PA| for a role	depends on the number of users assigned to a role	depends on |UA| and |PA| for all the roles	depends on experts' recommendations, role and user behaviour analysis	depends on the number of user behaviour patterns
Percentage of optimization	below 10%	below 10%	below 10%	not applicable	below 40%
Level of optimization	system level	system level	system level	user level	user level
Identification of obsolete UA	no	yes	no	no	yes
Refurbishment of UA	no	no	no	no	yes

the Recommended-RBAC, until the user's request or the system admin recommends a new set of roles. A detailed description of the flow of control is given below:

1. For a given sample dataset, user-permission assignments and its corresponding access privileges based on SRM algorithm are generated by the role miner.
2. Generated roles by the SRM algorithm are assigned to the users as well as permissions to build an RBAC system.
3. The logger stores the user behaviour pattern of all the

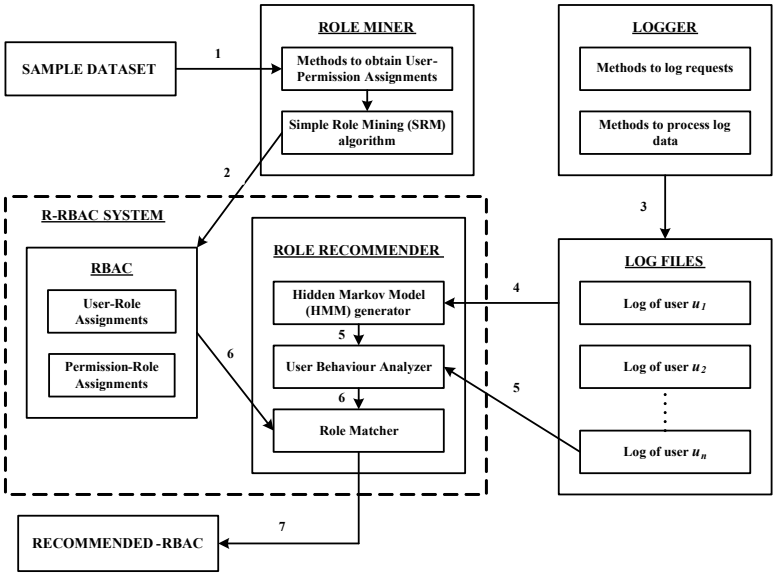

Figure 2: Proposed design of the R-RBAC System

users in the log files.

4. HMM generator generates the states for all the users using the corresponding user behaviour pattern.
5. To recommend the HMM state (i.e., set of permissions), user behaviour analyzer analysis the current usage of user behaviour pattern with the states generated by the HMM generator.
6. For a given user, the role matcher matches the recommended HMM states to the user privileges to generate the recommended role set.
7. The Recommended-RBAC is generated by assigning the recommended roles to the users as well as permissions.

Remark 1. *Before the construction of HMM, all the permissions should be accessed based on the privileges, and only the granted permissions are to be considered as user behaviour patterns.*

Remark 2. *Following the construction of HMM, users continue to access the permissions based on the privileges which are updated in the log files. At the time t, only the last k accessed permissions from the log files are considered for the role recommendation.*

Hosting R-RBAC in the Cloud: The hosting of the R-RBAC system by the CSP provides Role-Assignment-as-a-Service (RAaaS) for the deployed RBAC applications as depicted in the use-case diagram in Figure 3. Once the organization deploys the RBAC enabled applications on the cloud, an authorized user can access the permissions based on the sessions. Each such user sessions are assigned to roles, which are recommended by the R-RBAC to optimize the UA. Therefore, a service provided by the CSP to optimize the UA using the R-RBAC for the RBAC enabled cloud applications is referred to as Role-Assignment-as-a-Service. Such a service can also provide cost optimization for organizations that use built-in roles provided by the cloud (as an example- Azure resource manager) in deploying the applications.

3.2. Components of R-RBAC System

In this section, the components, i.e., RBAC and Role recommender, that are used to build the R-RBAC system are described.

3.2.1. RBAC

The NIST RBAC reference model defines different RBAC elements, RBAC assignments, and mapping functions [1]. The pictorial representation of the NIST RBAC system is shown in Figure 4. Let us define some of the

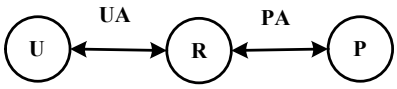

Figure 4: NIST RBAC [1]

RBAC elements, assignments and functions which are crucial for R-RBAC. Let U denote a set of users. Let R denote a set of roles, where each role is a job assigned to some users in an organization. Let P denote a set of permissions, where each permission is an approval to perform an operation on an object. Formally, this is expressed as $P = 2^{OP \times OB}$, where OP is the set of operations, and OB is the set of objects. Let **UA** denote a many-to-many mapping, which is user-to-role assignment relation, i.e., $\mathbf{UA} \subseteq U \times R$. Let **PA** denote a many-to-many mapping, which is a permission-to-role assignment relation, i.e., $\mathbf{PA} \subseteq P \times R$. Also, the **assigned_permissions:** $R \rightarrow 2^P$ is a mapping function from a role to a set of permissions [1]. So, for some $r \in R$, the **assigned_permissions(r)** = { p ∈ P | (p, r) ∈ **PA** }.

RBAC Configuration: Let **UP** denote a many-to-many mapping of user-to-permission assignment relation, i.e., $\mathbf{UP} \subseteq U \times P$. Also let us consider the function **SimpleRoleMining** which on input **UP**, outputs **UA** and **PA**, i.e., **SimpleRoleMining**: $U \times P \rightarrow \{ \mathbf{UA}, \mathbf{PA} \}$. The access control on RBAC can be defined as a *CheckAccess* function, which is responsible for the authorization process. The function is defined as *CheckAccess*: $U \times P \rightarrow P$. *CheckAccess* takes a user u and set of all permissions and returns a legitimate set of permissions pertaining to user u through the roles. Formally, $CheckAccess(u, P) = \{p : p \in P \wedge \forall r, (u, r) \in \mathbf{UA} \wedge (p, r) \in \mathbf{PA}\}$.

3.2.2. Role Recommender

In this section, the construction of the HMM (HMM generator), recommendation of the most suitable HMM state based on user behaviour patterns (User behaviour analyzer), and finally, matching the recommended state

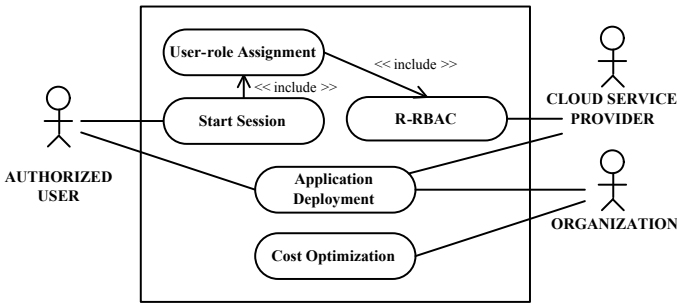

Figure 3: Hosting R-RBAC by the CSP providing Role-Assignment-as-a-Service

with RBAC (Role matcher) to generate an optimized UA at the user level is presented.

A. HMM Generator: The HMM is a stochastic model with probabilistic parameters such as *states*, *observations*, *transition probabilities* from one state to another, *emission probabilities* as well as *start probability* for each state. The observations are a set of possible permissions from each state. The transition probability is the probability of the occurrence of transition among the states. Emission probability is a property of the observed permissions at a particular time in a corresponding state of HMM. Start probability is an initial probability distribution over the states.

Here, the elements required for the construction of HMM is defined. Let BP_u denote a set of user behaviour pattern of $u \in U$ (refer Section 3.2.1), where $BP_u = \{p_1, p_2, \ldots, p_n\}$, such permission sequence is obtained from the log files (refer Figure 2). Let X_u denote a set of states of $u \in U$, where each state is associated with the previously encountered set of permissions along with one new permission from BP_u, i.e., $X_u = \{< p_1 >, < p_1, p_2 >, \ldots, < p_1, p_2, \ldots, p_n >\}$ and its corresponding observations are $\{\{p_1\}, \{p_1, p_2\}, \ldots, \{p_1, p_2, \ldots, p_n\}\}$. Once the state is generated, it is possible to find the number of times a particular permission has been granted from the state (*emission count map*). Additionally, the total number of granted permissions from the state (*frequency of occurrence of the state*) can also be counted. Further, in HMM, there exists a transition from one state to all the remaining states along with a self transition. Further, the probability of such transitions in HMM is given by, transition probability, $t_{ij} = p(x_j \mid x_i)$. Since the user needs to retain or change the state dynamically based on the business requirements, the probability of transition to all states is maintained at the same level all the time. Therefore, the transition probability from one state to any of the remaining n states is given by $\frac{1}{n}$. The probability of observed behaviour of permission p_k at time t when the user is in the state x_j is given by emission probability $e_{jk} = p(p_k \mid x_j)$. Then the number of times the permission is requested and granted at state x_j by frequency of occurrence of the state x_j is calculated. The start probability, π, is kept constant across all the states, and hence, was not considered in further calculations. Therefore, HMM was considered for user u as 4-tuple, $HMM = (States, Observations, Transition\ Probability, Emission\ Probability)$. The detailed construction of the HMM generator is shown in Algorithm 1.

Discussion on HMM Generator Algorithm. The user behaviour pattern, *BP*, is collected and given as input. Based on the input, states are generated for each user. States are a unique set of permissions and are observed/emitted at each state. Based on the occurrence of permissions in a state, the emission probability is calculated, and the transition probability is constant across all the states, which are calculated based on the total number of states for a given user. Therefore, the algorithm will continue to execute until the states and its probabilistic parameters are generated for all the users. An instance of Algorithm 1 depicting the characteristics of HMM states for a particular user is given in Table 4.

The number of states in the HMM for a given user is equal to the number of unique permissions in the user behaviour pattern. The time complexity of the HMM gener-

Algorithm 1: HMM Generator Algorithm

Input: Map<U, BP>
Output: HMM: Map<U, States, Observations, TransitionProbability, EmissionProbability>

State()
  /* Characteristics of HMM state */
  Set<P> EmittedPermission ← ∅
  FrequencyofOccurenceoftheState ← 0
  EmissionCountMap ← Map<EmittedPermission, Count>
for *each* $u \in U$ **do**
  State CurrentState ← ∅
  Set<State> AllStates ← ∅, Set<P> PermissionSet ← ∅
  for *each* $p \in BP_u$ **do**
    if $p \in$ *CurrentState.EmittedPermission* **then**
      update(CurrentState)
    end
    else
      State NewState.EmittedPermission ← PermissionSet.add(p)
      update(NewState)
      CurrentState← NewState
      AllStates ← AllStates.add(CurrentState)
    end
  end
  Calculate the *EmissionProbability* and *TransitionProbability* for the *AllStates* of the user *u*
  HMM ← Map<u, AllStates, EmittedPermission, TransitionProbability, EmissionProbability>
end
return HMM
void **calculateEmissionProbabilities()**
  $e_{jk} \leftarrow \frac{EmissionCountMap.get(EmittedPermission)}{FrequencyofOccurenceoftheState}$
void **calculateTransitionProbabilities()**
  $t_{ij} \leftarrow \frac{1}{AllStates.size()}$

ator for a single user is $O(nm)$, where n is the number of user behaviour patterns with m unique permissions. Additionally, the required time for calculating emission probability as well as transition probability is $O(m^2)$. Since the transition probability is constant, the time complexity to calculate transition probability is $O(1)$.

B. User Behaviour Analyzer: Here the most suitable HMM state is recommended for user u at time t based on BP_u^k. Based on HMM generator (refer Algorithm 1), the states and its observations for user u having n number of user behaviour patterns with m unique permissions are given below:
States, $X_u = \{< p_1 >, < p_1, p_2 >, \ldots, < p_1, p_2, \ldots, p_m >\}$
Observations, $O_u = \{\{p_1\}, \{p_1, p_2\}, \ldots, \{p_1, p_2, \ldots, p_m\}\}$
To model many observations, subindices represents time slice as a dynamic Bayesian network as shown in Figure 5. Every recommended state x at time t denoted as $x(t)$

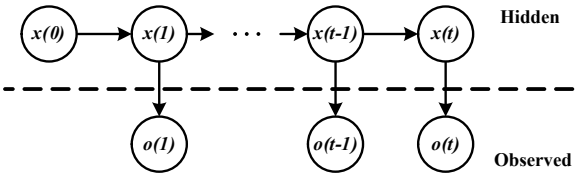

Figure 5: A dynamic Bayesian network specifying conditional independence relations for an HMM [25]

depends only on state at time $t-1$, i.e., $x(t-1)$ and observation of permissions at time t, i.e., $o(t)$ depends only on $x(t)$. Now, that BP_u^k has been observed at time t and assumed that user u was in the state $x(t-1) = x_m$, the recommended state $x(t)$ needs to be found, given $o(t) = BP_u^k$, $x(t-1) = x_m$.
It is also known that the probability of $x(t-1)$ denoted as $p(x(t-1))$ is equal to 1, because the state at the time $(t-1)$ already occurred; therefore,

$$p(x(t-1), x(t), o(t)) = p(o(t)|x(t)) * p(x(t)|x(t-1)) * p(x(t-1)) \quad (1)$$

Substituting the suitable values in equation (1),

$$p(x_m, x(t), BP_u^k) = p(BP_u^k|x(t)) * p(x(t)|x_m) * 1 \quad (2)$$

So, the state which takes a value with the *maximum likelihood estimation* for the equation (3) needs to be computed.

$$x(t) = \underset{x_j \in \{x_1, x_2, \ldots, x_m\}}{\text{argmax}} \left[\underbrace{p(BP_u^k|x_j)}_{\text{emission probability}} * \underbrace{p(x_j|x_m)}_{\text{transition probability}} \right] \quad (3)$$

An HMM state having *maximum likelihood estimation* at time t is called recommended state. Such a state is mapped to the PA (permission-role assignments), which will recommend the roles to optimize the UA. Therefore, state x_j having the maximum product value of transition and emission probability is selected as the most recommended state. Since transition probability is constant across all the states for a given user, the calculation can be reduced as shown below:

$$x(t) = \underset{x_j \in \{x_1, x_2, \ldots, x_n\}}{\text{argmax}} \left[\underbrace{p(BP_u^k|x_j)}_{\text{emission probability}} \right] \quad (4)$$

Such a recommended state needs to update the emission probability based on BP_u^k. The classes and methods are respectively stored in a package called *Analyzer*.

Discussion on Analyzer. Depending on the set of permissions requested, each user is recommended to be placed in a most suitable HMM state. Each state is associated with the last k permissions requested by user u given as BP_u^k. Thus, if the user has requested the permission sequence $p_1, \ldots, p_j$, they will be located in the state having maximum emission probability value for the corresponding permission sequences, observing *maximum likelihood estimation*. This is modelled using a dynamic Bayesian network specifying conditional independence relations for an HMM. Before closing this section, Table 2 provides the list of notations discussed in this section.

Table 2: Summary of notations representing user u

Notation	Description
BP_u	Permission sequence from the log files, i.e., user behaviour pattern
BP_u^k	Last k permission sequence from the log files i.e., current usage of user behaviour pattern
X_u	Set of HMM states
O_u	Set of observed permissions at each state
$x(t)$	Identified state at time t
$o(t)$	Observed permissions at time t
$p(x(t)\|x(t-1))$	Probability of transition between states at time slices $(t-1)$ to t, i.e., transition probability
$p(o(t)\|x(t))$	Probability of observed permissions at state $x(t)$, i.e., emission probability
p_i	Instance of permission
x_j	An instance of an HMM state
$< .. >$	Set of unique values
$\{..\}$	Set of values

C. Role Matcher: Here the roles are recommended by matching the recommended state with the permission-role assignments in RBAC system. Such recommended roles are assigned to users to generate an optimized UA in the Recommended-RBAC.

Recommended-RBAC Configuration: Let $\mathbb{R}$ be the recommended set of roles, i.e., $\mathbb{R} \subseteq R$ (refer Section 3.2.1). Therefore, the Recommended-RBAC is denoted as $\mathbb{RBAC}$, which is obtained by assigning only the recommended roles $\mathbb{R}$ to the users as well as permissions, which are referred to as $\mathbb{UA}$ and $\mathbb{PA}$, i.e., $\mathbb{RBAC} \rightarrow \{\mathbb{UA}, \mathbb{PA}\}$. Thus, the $\mathbb{RBAC}$ system generated by R-RBAC remains similar to Figure 4, and the authorization process is also the same as RBAC (refer Section 3.2.1).
Before closing this section, Table 3 provides the list of notations discussed in this section.

Table 3: Summary of notations

Notation	Description
RBAC	Roles generated and assigned to users as well as permissions using SRM algorithm
R	Set of roles defined in RBAC
UA	User-role assignments in RBAC
$\mathbb{RBAC}$	Recommended-RBAC, i.e., roles assigned to users as well as permissions based on user behaviour pattern using R-RBAC. Also $\mathbb{RBAC} \subseteq$ RBAC
$\mathbb{R}$	Set of roles defined in $\mathbb{RBAC}$, i.e., $\mathbb{R} \subseteq R$
$\mathbb{UA}$	Optimized user-role assignments in $\mathbb{RBAC}$, i.e., $\mathbb{UA} \subseteq$ UA

Remark 3. *HMM, which is represented as a dynamic Bayesian network, is a generative and probabilistic model, unlike other statistical models of the sort of Maximum-entropy Markov model and Conditional Random Fields, which are discriminative models, especially in supervised learning [24]. The proposed model is most suitable for role recommendation under the existing statistical models because the roles are recommended based on the state, which is generated using the user behaviour patterns.*

Remark 4. *In HMM, unlike neural networks, identification of the appropriate state depends on the type of permissions rather than on the sequence of permissions. Therefore, the HMM model can recommend the appropriate state even if the user changes the pattern of his behaviour. Further, in HMM, unlike neural networks, the user can randomly migrate between the states based on the transition probability to recommend the state. Additionally, while choosing the HMM state, neither random search nor Bayesian optimization can be a good solution because the user's preferred state depends on current behaviour patterns (refer to equation 4). In the case of a neural network, the network is constructed based on user behaviour pattern, where the output of one hidden layer is an input to another hidden layer. After the construction of a network, the system cannot give an appropriate*

role recommendation, if the user changes the pattern of his behaviour.

4. R-RBAC Analysis

In this section, the integration of a role recommendation model is presented for the RBAC system (i.e., R-RBAC) to optimize the UA based on user behaviour patterns. Further, an illustrative example will show the optimization of UA as well as security analysis on the R-RBAC system. Finally, the usefulness of the R-RBAC system in various IT environments is presented.

4.1. Optimizing User-Role assignments in RBAC

Here, the role recommendation model is shown for the RBAC to optimize the UA at the user-level in the Recommended-RBAC based on BP^k, as shown in Algorithm 2. To understand how obsolete and refurbished UA is handled, the BP^k_u is considered with a set of granted and denied permissions, which are handled differently by referring it as *AccessPermission* and *MissingPermission* as shown in Algorithm 2. The permissions that are granted and currently not accessed by the user are over-assigned permissions, whose roles need to be revoked from the user depending on *AccessPermission*. The permissions that are not accessible, since its corresponding roles are not refurbished to the user, are said to be missing permissions. The roles pertaining to such missing permissions are refurbished depending on *MissingPermission*. For a given set of *AccessPermission*, the most suitable state for the user is recommended by *FindClosestState* based on the equation (4), i.e., the computation of the HMM likelihood, using the *Forward* algorithm. For each of the permissions in the set that are considered to be added to a user by role, it is checked whether they are typical for the recommended state, i.e., *permissionIsTypicalForState*, using a metric *LowestSignificantPermisssionProbability* (LSP). Therefore, if the obtained emission probability value of permission is greater than or equal to LSP, then such permissions are typical for the recommended state. The *FindClosestExtendedState* considers only the permissions within the scope of user privileges to recommend a suitable state using equation (4). Finally, using the recommended state (i.e., *NecessaryPrms*), the roles are recommended for the user based on PA_u in the RBAC by the *Role matcher* to obtain optimization in UA, i.e., *OptimizedUA*. Such *OptimizedUA* is used to obtain Recommended-RBAC, as shown in Algorithm 2.

Algorithm 2: Optimizing user-role assignments in RBAC

```
   Input: BP^k
   Output: RecommendedRoles: Set<R>, Recommended-RBAC:
           Map<UA, PA>
 1 RBAC: Map<UA, PA>
 2 RecommendedState: Set<P>
 3 Recommended-RBAC ← RBAC
 4 LowestSignificantPermisssionProbability ← δ
 5 for each u ∈ U do
 6     {AccessPermission, MissingPermission} ← BP^k_u
 7     if AccessPermission then
 8         RecommendedState ←
               Analyzer.FindClosestState(AccessPermission)
           /* User behaviour analyzer                    */
 9         if RecommendedState ≠ NULL then
10             for each p ∈ RecommendedState do
11                 if permissionIsTypicalForState(p) then
12                     Set<P> NecessaryPrms.add(p)
13                 end
14             end
15         end
16     end
17     if MissingPermission then
18         RecommendedState ← Ana-
             lyzer.FindClosestExtendedState(MissingPermission)
19         if RecommendedState ≠ NULL then
20             NecessaryPrms.add(RecommendedState)
21         end
22     end
       /* Role matcher                                   */
23     RecommendedRoles ← Map<PA_u, 2^|NecessaryPrms| >
24     OptimizedUA ← Map <U, RecommendedRoles>
25     PA_u ← Map <P, RecommendedRoles>
26     Recommended-RBAC ← Map<OptimizedUA, PA_u >
27 end
```

Discussion on Optimizing User-Role Assignments in RBAC. The algorithm takes last k permissions from each user, i.e., BP^k_u, as an input. Such input is segregated based on granted and denied permissions which are further referred to as *AccessPermission* and *MissingPermission* respectively, based on currently assigned roles. The *Analyzer* recommends the HMM states based on such permission sets. Finally, the *Role matcher* performs in two phases: 1) recommended states are mapped based on its privileges in the RBAC to recommend the roles, and 2) a

user is assigned only with the recommended roles, which will generate optimized UA in the Recommended-RBAC. The algorithm will continue to execute until the optimized UA is generated for all the users based on BP^k.

4.2. An Illustrative Example

In this section, the R-RBAC is illustrated by optimizing the UA in RBAC with the help of a toy example, as shown in Figure 1. With reference to Figure 1, the following log file *ToyExample.log* containing user behaviour pattern of user u_2, i.e., $BP_{u_2} = \{p_1, p_1, p_3, p_3, p_1, p_2, p_3, p_2, p_2, p_2\}$ is considered, as given in Figure 6. Additionally, the log file also shows the permissions access status with respect to date and time. The log file *ToyExample.log* is referred to in this paper to show the role recommendation and optimization in the UA.

Table 4 shows the HMM model generated for user u_2 based on the user behaviour pattern given in Figure 6. The transition probability is given by the Markov matrix, where a transition between any two states is equal to 0.33. With reference to Table 4, the emission probability is given by

$$B = (e_{jk}) = \begin{array}{c} \\ p_1 \\ p_3 \\ p_2 \end{array} \begin{array}{c} \begin{array}{ccc} x_1 & x_2 & x_3 \end{array} \\ \begin{bmatrix} 1 & 0.33 & 0 \\ NA & 0.67 & 0.2 \\ NA & NA & 0.8 \end{bmatrix} \end{array}$$

Suppose the current usage of user behaviour pattern of user u_2 is observed, at time t as "p_2, p_3, p_2, p_3, p_2", where all the permissions in a given sequence are granted. If it is found that at time $(t-1)$ user u_2 was in state

Date (MDY)	Time	Permission	Status
10-01-2019	17:32:42.768	p_1	granted
10-01-2019	17:32:42.768	p_1	granted
10-01-2019	17:32:42.768	p_3	granted
10-01-2019	17:32:42.768	p_3	granted
10-01-2019	17:32:42.768	p_1	granted
10-01-2019	17:32:42.768	p_2	granted
10-01-2019	17:32:42.768	p_3	granted
10-01-2019	17:32:42.768	p_2	granted
10-01-2019	17:32:42.768	p_2	granted
10-01-2019	17:32:42.768	p_2	granted

Figure 6: System usage log containing the user behaviour pattern for user u_2 (*ToyExample.log*)

Table 4: Characteristics of HMM states for the user u_2

State (X_{u_2})	Description of the State (PermissionSet)	Frequency of occurrence of the state	Emission Count Map	
			Emitted Permission	Count
x_1	$< p_1 >$	1	p_1	1
x_1	$< p_1 >$	2	p_1	2
x_2	$< p_1, p_3 >$	1	p_1	0
			p_3	1
x_2	$< p_1, p_3 >$	2	p_1	0
			p_3	2
x_2	$< p_1, p_3 >$	3	p_1	1
			p_3	2
x_3	$< p_1, p_3, p_2 >$	1	p_1	0
			p_3	0
			p_2	1
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
x_3	$< p_1, p_3, p_2 >$	5	p_1	0
			p_3	1
			p_2	4

x_3, then which is the most suitable state for user u_2 as recommended by the model based on the observation sequence? Given $o(t) = BP^k_{u_2} = \{p_2, p_3, p_2, p_3, p_2\}$, $k = 5$, $AccessPermission = \{p_2, p_3, p_2, p_3, p_2\}$, $MissingPermission = \emptyset$, and $x(t-1) = x_3$, find the recommended state x_t.

Before optimization, there is $|R_{u_2}| = 2$, i.e., $\{r_1, r_2\}$ and $|UA| = 2$ (refer to Figure 1). Substituting the above values in equation (4),

$$x(t) = \underset{x_j \in \{x_1, x_2, x_3\}}{\text{argmax}} \left[\underbrace{P(BP^k_{u_2} | x_j)}_{\text{emission probability}} \right] = x_3$$

Here, the states containing *NA* (not applicable) are not taken into consideration while recommending the state. Therefore, based on the obtained value, state x_3 is the most suitable and recommended state for the given observation sequence $BP^k_{u_2}$. Finally, for the recommended state (x_3), the emission probability is updated as shown below:

$$B = (e_{jk}) = \begin{array}{c} \\ p_1 \\ p_3 \\ p_2 \end{array} \begin{array}{c} \begin{array}{ccc} x_1 & x_2 & x_3 \end{array} \\ \begin{bmatrix} 1 & 0.33 & 0 \\ NA & 0.67 & 0.3 \\ NA & NA & 0.7 \end{bmatrix} \end{array}$$

Once the state is recommended as x_3, it is to be checked whether all the permissions are typical for the state. So it

is assumed that $LSP = 0.01$, i.e., a heuristic value for the toy example. From Table 4, it can be seen that the permission set for state x_3 is $< p_1, p_2, p_3 >$. Since the emission probability value of $p_1 < LSP$, permission p_1 is not typical for the state. Therefore, the role recommended by R-RBAC is only r_2. Further, the Recommended-RBAC is updated with a new recommended role r_2. After optimization by R-RBAC, it can be observed that $|R_{u_2}| = 1$, i.e., r_2 and $|UA| = 1$.

4.3. Security Analysis

In this section, security analysis related to users' accessibility on permissions through roles using the R-RBAC system is elaborated.

It is known that role recommendation plays a crucial role in R-RBAC for optimizing the UA. So, the recommended roles should give access to only those permissions, which are in the user's scope as defined in the RBAC while optimizing the UA based on current usage of user behaviour pattern (i.e., BP_u^k). Permissions within the scope of user's privilege, is referred to as *accessible* permissions. Similarly, the recommended roles should not give access to those permissions, which are not in the user's scope as defined in the RBAC, which is referred to as *inaccessible* permissions. Table 5 signifies the possible outcomes of the R-RBAC based on its role recommendation for the granted and denied permissions.

As long as the user is accessing the permissions within his scope of privileges and predefined period, he can influence the stochastic model; however, this will not be a security issue. Further security issues are discussed in Theorems 1 and 2 when the user accesses out of scope privileges by colluding with a the malicious entity.

Definition 1. *R-RBAC is said to be secure if it recommends only roles within the scope of user privileges as defined in the RBAC system.*

In Theorem 1, the behaviour of a malicious user (i.e., adversary) is studied while accessing the permissions as well as the adversary's scope of accessible permissions on recommending the roles.

Theorem 1. *For any polynomial time adversary attempting to access finitely many permissions, the R-RBAC deterministically distinguishes the scope of privileges under AccessPermission and MissingPermission to recommend the roles.*

Proof. It is known that *AccessPermission*, i.e., granted permissions, is always within the scope of user privileges. So here only the *MissingPermission* is considered, i.e., denied permissions, where there is a possibility of recommending out of the scope privileges.

Let us consider all the denied permissions as missing permissions during the role recommendation by the R-RBAC system. In the R-RBAC, let ρ be the set of all permissions that are attempted to access by adversary u, but are denied based on the assignment of the recommended role set $\mathbb{R}_u$ in $\mathbb{RBAC}$, which are now considered as missing permissions and let $\rho = \{p_i, p_j\}$. In RBAC, let $R_u = \{r_i\}$ and also let adversary u access the permission $P(r_i) = p_i$ through the assigned role set R_u, i.e., *CheckAccess*$(u, P) = P(r_i)$. So, $P(r_i) = \{p : p \in P, (u, r_i) \in \mathbf{UA} \wedge (p, r_i) \in \mathbf{PA}\}$. So it is known that adversary u cannot access the permission p_j through the assigned role set R_u. During the recommendation process, based on the *CheckAccess* in RBAC, only role r_i is added to the recommended role set $\mathbb{R}_u$. It is noted that the role set $\mathbb{R}_u$ is now restored for the adversary u to access the missing permission p_i in the $\mathbb{RBAC}$. Hence, before recommending the role set $\mathbb{R}_u$, R-RBAC verifies with RBAC using the authorization process. Therefore, R-RBAC recommends only the roles, which are in the scope of user privileges in $\mathbb{RBAC}$. Hence, the role recommended by R-RBAC is secure and in line with RBAC.

Remark 5. *It is known that, in RBAC, each permission is an approval to perform an operation on an object. Therefore, here permission is considered as certain operations on the user's data (i.e., object). From Theorem 1, it is clear that the data confidentiality of the user is retained.*

Therefore, from Theorem 1, the R-RBAC system is secure under Definition 1, which ensures high *data confidentiality* by recommending only the roles within the scope of user privileges.

In the next lemma, we study the probability of success in accessing the permissions.

Lemma 1. *Let BP_p be the observed user behaviour pattern at time t, i.e., $o(t) = BP_p$. Also, let $BP_p = g + d$,*

Table 5: Possible outcomes of R-RBAC based on its role recommendation for user u

Access control status	**Scope of the user's privilege (*accessible* permissions)**		**Not in the scope of user's privilege (*inaccessible* permissions)**
	Belong to BP_u^k	**Do not belong to BP_u^k**	
Granted permissions	necessary permissions, so retain its UA	over-assigned permissions, so revoke its UA	not applicable
Denied permissions	missing permissions, so refurbish its UA	missing permissions, but do not refurbish its UA	missing permissions, but do not refurbish its UA

where $'g'$ corresponds to the number of granted permissions and $'d'$ corresponds to the number of denied permissions which are out of the scope of the user privileges. Then $Pr[granted\ permissions] = \frac{g}{BP_p}$.

Proof. Since the role recommendation depends only on the list of observed user behaviour pattern, the probability of success in accessing the permissions is $\frac{g}{BP_p}$. □

In Theorem 2, the effect of accessing out of scope privileges on the availability of the existing privileges, i.e., roles is discussed.

Theorem 2. *User privileges are suppressed if the users are trying to access a large number of permissions that are out of the scope of the user privileges.*

Proof. In R-RBAC, at the time $(t-1)$, let the number of roles in the recommended role set $\mathbb{R}$ be γ, i.e., $|\mathbb{R}| = \gamma$ and the number of user-role assignment $\mathbb{UA}$ based on the $\mathbb{R}$ in $\mathbb{RBAC}$ be ψ, i.e., $|\mathbb{UA}| = \psi$. It is noted that $\mathbb{UA}$, i.e., a map from users to roles is dependent on the number of roles and reduction in $|\mathbb{R}|$ amounts to reduce $|\mathbb{UA}|$ also. Let BP_p be the observed user behaviour pattern at time t, i.e., $o(t) = BP_p = g + d$, where $'g'$ corresponds to number of granted permissions and $'d'$ corresponds to number of denied permissions which are out of the scope of the user privileges and $g << d$. Since from Theorem 1 only roles within the scope of user privileges are recommended. We know $Pr[suppressed\ permissions] = 1 - Pr[granted\ permissions]$. Also from Lemma 1, $Pr[granted\ permissions] = \frac{g}{BP_p}$. So, $Pr[suppressed\ permissions] = \frac{d}{BP_p}$. Since $g << d$, $Pr[granted\ permissions] < Pr[suppressed\ permissions]$. At time t only a small share of BP_p, i.e., $'g'$ will contribute to $\mathbb{R}$ compared with its state at time $(t-1)$, i.e., $|\mathbb{R}| << \gamma$. So the number of roles assigned to the users $\mathbb{UA}$ in the $\mathbb{RBAC}$ is also less than ψ, i.e., $|\mathbb{UA}| << \psi$, since $g << d$. Hence, the user privileges in terms of roles at time t are suppressed. □

Remark 6. *From Lemma 1, it is worth noticing that when $g > d$, the success rate is higher. When $g < d$, the success rate is less, which is a hint for the user to revisit the log, so has to ensure that suppressed permissions may not take over. One realistic attack scenario relates to the adversarial attempt to suppress a user's permission. From Lemma 1, when $g = d = \frac{1}{2}$, the entropy of the system is highest. In this attack, the adversary can contribute in accumulating $'d'$ by injecting fraudulent attempts to access so that in the log, $d \approx g$. In such a case when $'g'$ access is denied, the user will have less clue as to what takes over the situation - the suppressed permissions or the granted permissions.*

Remark 7. *From Theorem 2, users trying for some unanticipated situations by accessing the unauthorized roles reduce the access privileges, which in turn affects the availability of the existing role set.*

4.4. Usefulness in IT Environments

Automated Extension of Access: The role recommendation model can provide an effective solution by providing automated access in RBAC applications. Employees are given access to the laboratory for a certain period of time; after that the employees have to request for extension of privilege, i.e. roles. The manual updation for extension of access by the security administrator is time consuming and increases administrative workload. Such work can be automated with the help of the R-RBAC system, which results in faster and efficient access.

Security at Hypervisor Level: In a virtualized environment, the hypervisor is a single point of failure for all the

virtual machines running on it. Using a compromized hypervisor, the attacker can launch a Denial of Service (DoS) attack across a collection of virtual machines by increasing its resource usage. This makes machine resources unavailable, i.e., intended users cannot access the information deployed in the virtual machines [28]. Restricting the resource usage to specific limits based on the privilege (i.e., "roles") assigned for a virtual machine, is a security measure against hypervisor based DoS attacks. Based on the requirements, the limits on resource usage for a virtual machine can be changed by dynamically maintaining the roles for virtual machines with the help of the R-RBAC system.

5. Experimental Results

In this section, the experimental set-up is reported by developing a role recommendation model for the RBAC. The performance analysis of the proposed system is also provided using the Amazon access sample dataset [8] and on the generated synthetic datasets, which was executed based on the developed test cases. The effective experimental results for the proposed model can be shown in terms of *speed* and *size* metrics using the results obtained from the optimal solution in UA. Since optimizing UA by satisfying the least privilege principle is proved to be NP-hard, we have compared the above metrics with the existing models which provides good approximation to the optimal solution in contrast to the state-of-the-art. Additionally, the similarity between the roles recommended by the proposed model and the required optimal roles can also be analyzed to show the effective experimental results. The implementation is done on Intel Core(TM) $i5-7500$ with 16 GB RAM using Java in the Windows platform.

To test the system, the user's log dataset of the system access over certain time was required. Secondly, an RBAC structure was required. The role mining algorithm can also be used to design a set of roles based on user-permission assignments. Since the company log data is generally considered as sensitive and very difficult to obtain, an anonymized dataset provided in [8] was used for the experiments.

5.1. Creation of RBAC on Amazon Access Sample Dataset [8]

The Amazon access sample dataset [8], is an anonymized sample of access rights provisioned in Amazon's Infosec and does not include all the logged access requests. It also does not include user-permission assignments, so this too had to be stipulated from the given log data. The attributes of the data set are *action*, *target name*, *login*, *request date*, and *authorization date*. *Action* refers to whether the permission is granted or denied, *target name* represents the permission, and *login* is considered as user credentials. *Request date* and *authorization date* is a timestamp that denotes the date and time on which the user requested and authorized the permission. Therefore, for the Amazon access sample dataset [8], the SRM algorithm was applied and the statistics are shown in Table 6. Further, the implemented SRM algorithm was validated using nine real-world datasets [29], which confirmed that the results are matching as summarized in [13].

Table 6: Statistics for Amazon access sample dataset [8] using SRM algorithm

Input/Output	**Values**
Total number of permission access status	716,063
Total number of permission access status- granted	705,152
Total number of permission access status- deny	10,911
Total number of user, |U|	5,885
Total number of permissions, |P|	6,451
Generated |UP|	144,435
Generated |UA|	39,389
Generated |PA|	48,845
Generated |R|	5,042
Maximum number of permissions for a user	125
Maximum number of roles for a user	46
Maximum number of permissions in a role	83

5.2. Automation of HMM Generator

In this section, the automated process of generating the HMM is discussed, which helps in role recommendation for the users. The framework was developed to generate the HMM based on user behaviour pattern for each user as shown in Figure 7. Further, the automation framework accesses all the authorized permissions of the users such that all possible states are registered with the HMM generator. The permission requester spawns parallel requests

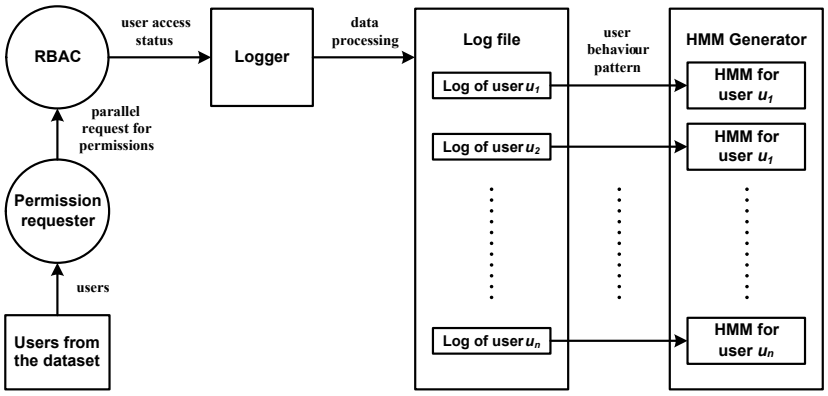

Figure 7: Automation framework to generate the HMM for role recommendation

for maximum of 5 permissions based on the RBAC from each user every 1 minute to record the user's permission access status in the logger. The log data contains permissions, which are processed to classify the data based on the user. Therefore for each user, HMM is generated based on the log file as specified in Algorithm 1.

Remark 8. *In all the experiments, to recommend the role at time t, the last state of the HMM is considered to be at time* $(t-1)$*. Therefore, the role is recommended based on the observation of permissions at time t which is considered as* BP^k*, i.e.,* $o(t) = BP^k$*.*

5.3. Performance Analysis of R-RBAC

In this section, the performance evaluation of R-RBAC in recommending the roles is shown. In the evaluation, two different metrics were considered, namely, *speed* and *size*. *Speed* and *size* are defined as processing time to recommend the roles and percentage of recommended roles respectively, by the model against the variation of the considered parameters. Further, the role recommendation model was compared with the existing models.

Performance of R-RBAC in Recommending the Role: In the next lemma, a complexity analysis of role recommendation in the R-RBAC is provided.

Lemma 2. *Using R-RBAC, the number of operations for recommending the role set* $\mathbb{R}_u$ *among role set* R_u *in RBAC for a given user u based on current usage of user behaviour pattern* BP_u^k *at time t, among HMM states* X_u *with total set of permissions* λ *is* $\theta(k * |\lambda|) + O(PA_u * (|\Upsilon| * ln2))$*, where* Υ *is the set of permissions in the recommended HMM state* $x \in X_u$ *and* PA_u *is the permission-role assignments for user u.*

Proof. *Time complexity to recommend HMM state:* Here BP_u^k recommends one of the HMM state, $x \in X_u$ with permission set Υ at the time t, based on the dynamic Bayesian network using forward algorithm. So the time required to recommend the HMM state x at time t, i.e., $x(t)$, with the total set of permissions λ in the HMM states, for the given BP_u^k is $\theta(k * |\lambda|)$.
Time complexity to recommend the roles for AccessPermission/MissingPermission: Roles are recommended based on the permission set Υ on the recommended state $x \in X_u$ with the possible number of combinations $2^{|\Upsilon|}$. Since this combination of $2^{|\Upsilon|}$ is done in lexicographic order, without the loss of generality, binary search can be applied. Therefore, the time complexity to recommend the roles for *AccessPermission/MissingPermission* is $O(\mathrm{PA}_u * ln(2^{|\Upsilon|})) = O(\mathrm{PA}_u * (|\Upsilon| * ln2))$.

Speed of R-RBAC on Amazon Access Sample Dataset [8]: Here, the *speed*, i.e., the time taken by the R-RBAC on recommending the roles for a single user is reported, where different number of permissions are observed at time t, i.e., $o(t)$. Initially, the HMM model is automated (refer Figure 7) for Table 6. The experiment is conducted on the Amazon access sample dataset [8] to recommend the roles on the varying number of last k accessed permissions at time t. Therefore, $o(t) = BP^k$ is

considered, where $k \in \{25, 50, 75, 100\}$. In Figure 8, the average time required among users for role recommendation (along Y-axis) for the given number of current usage of user behaviour patterns (along X-axis) is plotted. Figure 8 reflects a linear growth on *speed* from 22 to 92

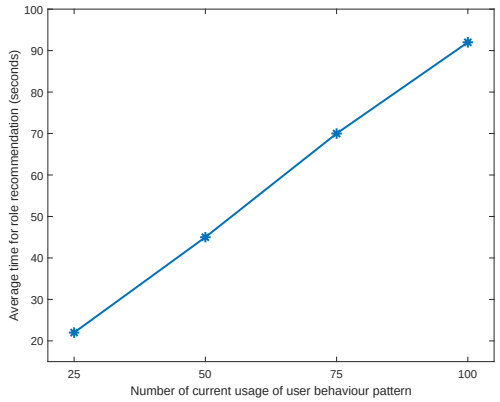

Figure 8: *Speed* of role recommendation on varying current usage of user behaviour patterns

seconds, with an increase in the number of current usage of user behaviour patterns from 25 to 100.

Size of R-RBAC on Amazon Access Sample Dataset [8]: Here, the *size*, i.e., the percentage of recommended roles by R-RBAC is reported, where four different test cases can be observed based on the RBAC (refer Table 6) at time t as described below:

- **Test case 1**: Access all the permissions assigned to roles for a given user.
- **Test case 2**: Do not access permissions pertaining to 50% roles for a given user, so that it appears like the user potentially did not need such obsolete UA.
- **Test case 3**: Add denied request in the log for a given user, so that it appears like the user is requesting to refurbish for 10% UA for such missing permissions.
- **Test case 4**: Access 50% of the roles that are not in the scope of user privileges.

Let β_i be the number of accessed permissions in the **Test case i**, where $i \in \{1, 2, 3, 4\}$. Therefore, the experimentation for the role recommendation is done by maintaining the principle of least privilege based on the test cases which is now considered as the set of last k accessed permissions at time t. Based on the above observations, $o(t) = BP^k$, where $k \in \{\beta_1, \beta_2, \beta_3, \beta_4\}$. Similar to Figure 7, a test automation framework was also generated for the execution of four different test cases. Applying the test cases to the HMM model generated from Table 6, the following results were derived.

In test case 1, all the roles were maintained for the users and there was no change in the roles recommended by the R-RBAC. In test case 2, it was found that the average role recommended by the R-RBAC for each user was 57% of the roles when 50% of the UA was obsolete. In test case 3, all the UA pertaining to missing permissions was refurbished based on corresponding role assignments in the RBAC. It was found that the average role recommended by the R-RBAC for each user was to restore 28% of the roles when 10% of UA was needed to be refurbished. If the missing permissions for any user were assigned with more than 10% of UA, then it was recommended to reconstruct the Recommended-RBAC for that corresponding user. In test case 4, it was clear from Theorem 2 that in R-RBAC, user privileges are suppressed if users access privileges not in their scope. So, the average role recommended by the R-RBAC for each user was 57% when the log contained 50% out of scope privileges, which is the same as test case 2.

5.4. Comparison with Other Models

In this section, the performance evaluation of the proposed system compared with LPUAP [6] in terms of *size* and *speed* is reported using synthetic datasets. This is because LPUAP [6] is the only model which at least identifies the obsolete UA to optimize the UA. Additionally, comparison is also presented with the model proposed in [21].

Size of R-RBAC and LPUAP [6] on Synthetic Datasets: Here, the *size*, i.e., percentage of recommended roles by the R-RBAC and the LPUAP [6] for the generated synthetic datasets is reported. In [6], GA-UADel and GA-UAAdd algorithm are used for role adjustments, i.e., to delete and add roles respectively from the system.

The approach of generating the synthetic datasets using test data creator algorithm (or training solution) is sug-

gested to evaluate the performance of role mining algorithms [9]. Further, the training solution used here is the same as specified in [9] to generate synthetic datasets. Towards this, two synthetic datasets were generated and are presented in Table 7 and Table 8 to understand the performance of the R-RBAC on role recommendation. Since the algorithm, i.e., test data creator is randomized, it is executed five times on each particular set of parameters to generate the synthetic datasets. The proposed model was run on each of the created synthetic datasets. All the results obtained from the specific parameters were averaged over the five runs. Both the synthetic datasets consisted

Table 7: Users with a varying number of permissions for a constant number of roles

#U	#R	#P	#Maximum roles for a user	#Maximum permissions for a role
1000	100	400	20	6
1000	100	600	20	9
1000	100	800	20	12
1000	100	1000	20	15

Table 8: Users with a varying number of roles for a constant number of permissions

#U	#R	#P	#Maximum roles for a user	#Maximum permissions for a role
1000	100	500	20	10
1000	150	500	20	10
1000	200	500	20	10
1000	250	500	20	10

of five parameters, namely, total number of users (#U), total number of roles (#R), total number of permissions (#P), maximum number of roles for a user, and maximum number of permissions for a role. In the first synthetic dataset, number of permissions and maximum number of permissions for a role are varying parameters, while the other parameters are constant as shown in Table 7. In the second synthetic dataset, total number of roles is a varying parameter, while other parameters are constant as shown in Table 8. For the experimentation, $o(t) = BP^k$ was considered, where $k = \beta_2$. Once the RBAC was constructed for Table 7 and Table 8, the HMM model was automated (refer Figure 7) for the corresponding tables, and test case 2 was also automated. In Figure 9, the plot of the average percentage of roles recommended for the users (along Y-axis) for the given permissions/roles ratio (along X-axis) based on the values tabulated in Table 7. Figure 9 shows that for R-RBAC, the *size* varies

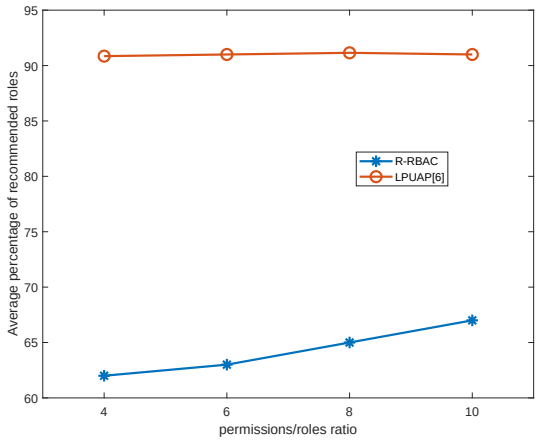

Figure 9: Comparison of *size* on varying number of permissions for a constant number of roles

from 62% to 67%, whereas for LPUAP [6], the *size* is almost a constant value of 91%, over permissions/roles ratio varying from 4 to 10. Similar to Figure 9, an experiment was done based on the values tabulated in Table 8, which showed a constantly recommended role as 63% and 91% for the R-RBAC and the LPUAP [6] respectively, over varying permissions/roles value from 2 to 5. Therefore, the above comparison reveals that the *size* efficiency of R-RBAC over LPUAP [6] is up to 32%.

Therefore from the above results, it can be observed that lesser the number of permissions assigned to each role, higher is the efficiency in recommending the roles. So, R-RBAC is most suitable for applications where roles are assigned with fewer numbers of permissions. The percentage of role recommendation is slightly higher than the optimal solution because along with the recommendation, the proposed model also predicts the roles which may help the user to access the permissions shortly.

Speed of R-RBAC and LPUAP [6] on Synthetic Dataset: Here, the *speed*, i.e., time taken by the R-RBAC

Table 9: Users with a varying maximum number of assigned roles

#U	#R	#P	#Maximum roles for a user	#Maximum permissions for a role
1000	100	300	4	5
1000	100	300	6	5
1000	100	300	8	5
1000	100	300	10	5

and the LPUAP [6] to recommend the roles based on the generated synthetic dataset is reported.

Experiments are conducted based on the synthetic dataset, where all the parameters are constant but with a varying maximum number of roles assigned for the users, as given in Table 9. With regards to R-RBAC, all the permissions based on the RBAC system are observed at time t, i.e., $o(t)$ and a set of last k accessed permissions at time t as 50 is maintained, since each user has a privilege of maximum 50 permissions (refer Table 9). In Figure 10, the average time taken to recommend the roles using the R-RBAC and the LPUAP [6] model (along Y-axis) on the varying number of roles assigned for the users (along X-axis) is plotted. Figure 10 shows that the R-RBAC takes

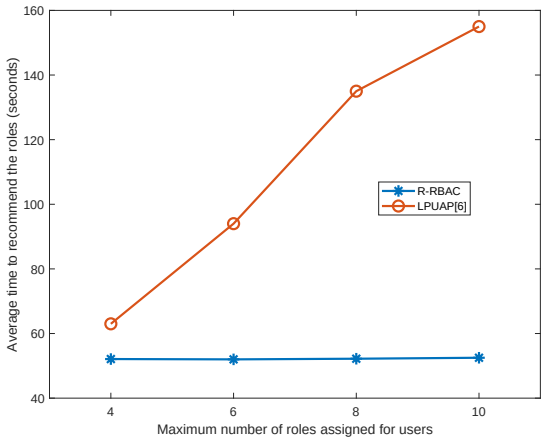

Figure 10: Comparison of *speed* on a varying maximum number of roles assigned for users

almost a constant time of 52 seconds to recommend the roles, whereas, in the case of LPAUP [6], the time varies from 63 to 155 seconds while the maximum number of roles assigned for users varies from 4 to 10. Based on the above experiment, the *speed* efficiency of the R-RBAC over the LPUAP [6] is up to 50%. Therefore, the R-RBAC is efficient in terms of *size* and *speed* because the roles are recommended at the user level, unlike the LPUPA [6] at the system level. So, the R-RBAC is most suitable for applications where role recommendation is needed at the user level to optimize the UA.

Comparison with the Model in [21]: In trust and reputation based RBAC system [21], the owners are service providers who share their resources, data, or services with the intended users in terms of eligible roles. Such eligibility of roles is based on their trust level. In this approach, a group with large number of users may collude with a malicious entity to either decrease or increase the other entities' reputation. Further, the solution proposed in [21] is ineffective, if the number of malicious entities in the RBAC system is more than fifty per cent. Additionally, if the user is found to be malicious, then his trust level towards the roles is decreased and the corresponding roles are revoked, if the user trust level depletes under the specified threshold of their roles. Therefore, by maintaining the required trust value, the user can use or misuse the system. Thus, the role recommendation cannot be trusted.

In the proposed system, the RBAC configuration was generated using the SRM algorithm in the first step. Further, in the second step, the roles were recommended based on the scope of user privileges defined in the first step for the three scenarios described in the introduction section. Users colluding with a malicious entity cannot influence the stochastic model for accessing the permissions which are not in the scope of the user's privilege (refer Section 4.3). Therefore in the R-RBAC, role recommendation can be trusted because the recommended roles are within user privileges (refer Theorem 1) and maintain the required security, while accessing out of scope privileges (refer Theorem 2).

6. Conclusion

Role recommendation at the user level is the most preferred and efficient solution for the optimization of user-role assignments (UA) in the RBAC. Towards this,

the Role recommender-RBAC (R-RBAC) was designed, which optimizes the UA, by recommending and assigning the most suitable roles. Such role recommendation was developed using a Hidden Markov Model and analyzed using the dynamic Bayesian network. While hosting the R-RBAC on the cloud platform, the CSP can provide Role-Assignment-as-a-Service, which optimizes the organizational operation cost. It has been shown that the R-RBAC is secure and in line with the RBAC system, and it suppresses user privileges while accessing out of scope privileges.

The experiment was conducted on the Amazon access sample dataset [8] and synthetic datasets by generating RBAC configuration using the Simple Role Mining algorithm. The experiment showed that the time for role recommendation varied linearly with the increase in user behaviour/access patterns. To evaluate the size of the recommended role set, four different test cases were considered. Such test cases are considered as the least privileges during role recommendation. It was shown that the R-RBAC recommends roles to optimize the UA 1) by retaining an average of 57% of roles when 50% of UA is identified as obsolete, 2) by restoring an average of 28% of roles when 10% of UA need refurbishment, and 3) by maintaining all necessary roles. Based on the experiments using synthetic datasets, the performance of the R-RBAC was found to be relatively good in terms of size and speed of optimization because optimizing UA was performed at the user level, unlike the existing models which perform UA optimization at the system level.

[1] R. Sandhu, D. F. Ferraiolo, D. R. Kuhn, The NIST Model for Role-Based Access Control: Towards a Unified Standard, Tech. rep., Laboratory for Information Security Technology (2000).

[2] B. Mitra, S. Sural, J. Vaidya, V. Atluri, A Survey of Role Mining, ACM Computing Surveys 48 (4) (2016) 50:1–50:37.

[3] D. Ferraiolo, R. Sandhu, S. Gavrila, R. Kuhn, Proposed NIST Standard for Role-Based Access Control, ACM Transactions on Information and System Security 4 (3) (2001) 224–274.

[4] J. Vaidya, V. Atluri, Q. Guo, The Role Mining Problem: Finding a Minimal Descriptive Set of Roles, in: Proceedings of 12th ACM Symposium on Access Control Models and Technologies, 2007.

[5] J. Vaidya, V. Atluri, Q. Guo, H. Lu, Edge-RMP: Minimizing administrative assignments for role-based access control, Journal of Computer Security 17 (2) (2009) 211–235.

[6] H. Huang, F. Shang, J. Liu, H. Du, Handling least privilege problem and role mining in RBAC, Journal of Combinatorial Optimization 30 (1) (2015) 63–68.

[7] H. Xia, M. Dawande, V. Mookerjee, Role Refinement in Access Control: Model and Analysis, INFORMS Journal on Computing 26 (4) (2014) 866–884.

[8] Amazon Access Samples Data Set, https://archive.ics.uci.edu/ml/datasets/Amazon+Access+Samples, UC Irvine Machine Learning Repository, (Accessed 29 September 2016).

[9] J. Vaidya, V. Atluri, J. Warner, Q. Guo, Role engineering via prioritized subset enumeration, IEEE Transactions on Dependable and Secure Computing 7 (3) (2010) 300–314.

[10] T. FitzMacken, Azure Resource Manager Overview, Microsoft Corporation, (Accessed 3 October 2018).

[11] J. Schlegelmilch, U. Steffens, Role mining with ORCA, in: Proceedings of 10th ACM Symposium on Access Control Models and Technologies, Sweden, 2005.

[12] A. Ene, W. Horne, N. Milosavljevic, P. Rao, R. Schreiber, R. E. Tarjan, Fast exact and heuristic methods for role minimization problems, in: Proceedings of 13th ACM Symposium on Access Control Models and Technologies, Colorado, 2008.

[13] C. Blundo, S. Cimato, A simple role mining algorithm, in: Proceedings of 25th ACM Symposium on Applied Computing, Switzerland, 2010.

[14] I. Molloy, H. Chen, T. Li, Q. Wang, N. Li, E. Bertino, S. Calo, J. Lobo, Mining roles with multiple objectives, ACM Transactions on Information and System Security 13 (4) (2010) 36:136:35.

[15] B. Mitra, S. Sural, V. Atluri, J. Vaidya, Toward mining of temporal roles, in: Proceedings of 27th international conference on Data and Applications Security and Privacy, New Jersey, 2013.

[16] W. Ye, R. Li, X. Gu, Y. Li, K. Wen, Role mining using answer set programming, Future Generation Computer Systems 55 (2016) 336–343.

[17] K. Rao, I. Ray, W. Asif, A. Nayak, M. Rajarajan, R-PEKS: RBAC Enabled PEKS for Secure Access of Cloud Data, IEEE Access 7 (2019) 133274–133289.

[18] M. Al-Kahtani, R. Sandhu, A model for attribute-based user-role assignment, in: Proceedings of 18th Annual Computer Security Applications Conference, Las Vegas, 2002.

[19] S. Chakraborty, I. Ray, TrustBAC: integrating trust relationships into the RBAC model for access control in open systems, in: Proceedings of 11th ACM symposium on Access Control Models and Technologies, California, 2006.

[20] M. Saffarian, Q. Tang, W. Jonker, P. Hartel, Dynamic User-Role Assignment in Remote Access Control, Tech. rep., University of Twente (2009).

[21] M. Ghafoorian, D. Abbasinezhad-Mood, H. Shakeri, A Thorough Trust and Reputation Based RBAC Model for Secure Data Storage in the Cloud, IEEE Transactions on Parallel and Distributed Systems 30 (4) (2019) 778–788.

[22] L. Zhou, C. Su, Z. Li, Z. Liu, G. Hancke, Automatic fine-grained access control in SCADA by machine learning, Future Generation Computer Systems 93 (2019) 548–559.

[23] M. A. Rahman, E. Al-Shaer, Automated Synthesis of Distributed Network Access Controls: A Formal Framework with Refinement, IEEE Transactions on Parallel and Distributed Systems 28 (2) (2017) 416–429.

[24] K. Murphy, Dynamic Bayesian Networks: Representation, Inference and Learning, Tech. rep., University of California, Berkeley (2002).

[25] Z. Ghahramani, An Introduction to Hidden Markov Models and Bayesian Networks, International Journal of Pattern Recognition and Artifical Intelligence 15 (1) (2001) 9–42.

[26] I. Molloy, N. Li, T. Li, J. Lobo, Z. Mao, Q. Wang, Evaluating Role Mining Algorithms, in: Proceedings of 14th ACM Symposium on Access Control Models and Technologies, Italy, 2009.

[27] F. Liang, G. Yunchuan, A Survey of Role Mining Methods in Role-Based Access Control System, in: Workshop on Web Technologies and Applications, China, 2014.

[28] S. Singh, Y. Jeong, J. Park, A survey on cloud computing security: Issues, threats, and solutions, Journal of Network and Computer Applications 75 (2016) 200–222.

[29] R. Schreiber. Datasets used for role mining experiments, http://www.hpl.hp.com/personal/Robert_Schreiber/, (Accessed 9 September 2015).

K. Rajesh Rao received the B.E. degree in computer science and engineering from Visvesvaraya Technological University, Belgaum, in 2008, and the M.Tech. degree in computer science and information security from the Manipal Institute of Technology (MIT), MAHE, Manipal, India, in 2012, where he is currently pursuing the Ph.D. degree. He is currently a Visiting Researcher with the Information Security Group, City, University of London, U.K., and also an Assistant Professor-Senior with the Department of Information and Communication Technology, MIT. His research interests include but are not limited to access control models, cloud security, and soft computing.

Ashalatha Nayak received the B.Tech. and M.Tech. degrees in computer science and engineering from Mangalore University, Karnataka, India, and the Ph.D. degree from the School of Information Technology, IIT Kharagpur, in the area of model-based testing. She is currently a Professor and the Head of the Department of Computer Science and Engineering, Manipal Institute of Technology, MAHE, Manipal, India. Her research interests include semantic web, software testing, intelligent agents, and cloud security.

Indranil Ghosh Ray received the B.Sc. degree (Hons.) in mathematics from Calcutta University, in 2000, the MCA degree from the University of Kalyani, in 2003, and the Ph.D. degree in computer science from the Indian Statistical Institute, in 2016. He worked in the software industry for six years as a Software Engineer and a Senior Software Engineer. He also had been a Research Associate with the Information Security Group, City, University of London, U.K., from 2016 to 2019. Currently, he has been a Research Fellow with the Department of Computer Science, University of Warwick, U.K., since June 2019. His research interest includes but is not limited to homomorphic encryption and its application in privacy preserving, cloud computing, searchable encryption, MDS codes and its applications in lightweight cryptography, and algebraic immunity of S-Boxes based on power mappings.

Yogachandran Rahulamathavan is a lecturer and a program director for M.Sc. Cyber Security and Big Data program at Loughborough University's London Campus in the UK. His research interest is on developing novel security protocols to advance machine learning techniques to solve complex privacy issues in emerging applications, e.g., patient's healthcare data sharing, biometric authentication systems, identity management in cloud, etc. He received a Ph.D. degree in Signal and Information Processing from Loughborough University in 2011 and then worked as information security researcher at City, University of London, between 2011 and 2016. Currently, he is coordinating UK-India project (worth of £200k) between Loughborough University London, IIT Kharagpur and City, University of London.

Muttukrishnan Rajarajan is currently a Professor of security engineering with the City, University of London, U.K., where he currently leads the Information Security Group. He is a Visiting Researcher with the British Telecommunication's Security Research and Innovation Laboratory. His research interests include privacy-preserving data analytics, cloud computing, the Internet of Things security, and wireless networks. He has published well over 300 articles and continues to be involved in the editorial boards and technical programme committees of several international security and privacy conferences and journals. He is an Advisory Board Member of the Institute of Information Security Professionals, U.K., and acts as an advisor to the U.K. Government's Identity Assurance Programme (Verify U.K.).